02 | 协议：怎么设计可扩展且向后兼容的协议？

2020-02-17 何小锋

讲述：张浩
时长 10:49　大小 9.92M

你好，我是何小锋。上一讲我分享了 RPC 原理，其核心是让我们像调用本地一样调用远程，帮助我们的应用层屏蔽远程调用的复杂性，使得我们可以更加方便地构建分布式系统。总结起来，其实就一个关键字：透明化。

接着上一讲的内容，我们再来聊聊 RPC 协议。

一提到协议，你最先想到的可能是 TCP 协议、UDP 协议等等，这些网络传输协议的实现在我看来有点晦涩难懂。虽然在 RPC 中我们也会用到这些协议，但这些协议更多的是我们上层应用是透明的，我们 RPC 在使用过程中并不太需要关注他们的细节。那我今天要讲的 RPC 协议到底是什么呢？

可能我举个例子，你立马就明白了。HTTP 协议是不是很熟悉（本讲里面所说的 HTTP 默认都是 1.X）？ 这应该是我们日常工作中用得最频繁的协议了，每天打开浏览器浏览的网页就是使用的 HTTP 协议。那 HTTP 协议跟 RPC 协议又有什么关系呢？看起来他俩好像不搭边，但他们有一个共性就是都属于应用层协议。

所以**我们今天要讲的 RPC 协议就是围绕应用层协议展开的。**我们可以先了解下 HTTP 协议，我们先看看它的协议格式是什么样子的。回想一下我们在浏览器里面输入一个 URL 会发生什么？抛开 DNS 解析暂且不谈，浏览器收到命令后会封装一个请求，并把请求发送到 DNS 解析出来的 IP 上，通过抓包工具我们可以抓到请求的数据包，如下图所示：

协议的作用

看完 HTTP 协议之后，你可能会有一个疑问，我们为什么需要协议这个东西呢？没有协议就不能通信吗？

我们知道只有二进制才能在网络中传输，所以 RPC 请求在发送到网络中之前，他需要把方法调用的请求参数转成二进制；转成二进制后，写入本地 Socket 中，然后被网卡发送到网络设备中。

但在传输过程中，RPC 并不会把请求参数的所有二进制数据整体一下子发送到对端机器上，中间可能会拆分成好几个数据包，也可能会合并其他请求的数据包（合并的前提是同一个 TCP 连接上的数据），至于怎么拆分合并，这其中的细节会涉及到系统参数配置和 TCP 窗口大小。对于服务提供方应用来说，他会从 TCP 通道里面收到很多的二进制数据，那这时候怎么识别出哪些二进制是第一个请求的呢？

这就好比让你读一篇没有标点符号的文章，你要怎么识别出每一句话到哪里结束呢？很简单啊，我们加上标点，完成断句就好了。

同理在 RPC 传输数据的时候，为了能准确地“断句”，我们也必须在应用发送请求的数据包里面加入“句号”，这样才能帮我们的接收方应用从数据流里面分割出正确的数据。这个数据包里面的句号就是消息的边界，用于标示请求数据的结束位置。举个具体例子，调用方发送 AB、CD、EF 3 个消息，如果没有边界的话，接收端就可能收到 ABCDEF 或者 ABC、DEF 这样的消息，这就会导致接收的语义跟发送的时候不一致了。

所以呢，为了避免语义不一致的事情发生，我们就需要在发送请求的时候设定一个边界，然后在收到请求的时候按照这个设定的边界进行数据分割。这个边界语义的表达，就是我们所说的协议。

如何设计协议?

理解了协议的作用，我们再来看看在 RPC 里面是怎么设计协议的。可能你会问：“前面你不是说了 HTTP 协议跟 RPC 都属于应用层协议，那有了现成的 HTTP 协议，为啥不直接用，还要为 RPC 设计私有协议呢？”

这还要从 RPC 的作用说起，相对于 HTTP 的用处，RPC 更多的是负责应用间的通信，所以性能要求相对更高。但 HTTP 协议的数据包大小相对请求数据本身要大很多，又需要加入很多无用的内容，比如换行符号、回车符等；还有一个更重要的原因是，HTTP 协议属于无状态协议，客户端无法对请求和响应进行关联，每次请求都需要重新建立连接，响应完成后再关闭连接。因此，对于要求高性能的 RPC 来说，HTTP 协议基本很难满足需求，所以 RPC 会选择设计更紧凑的私有协议。

那怎么设计一个私有 RPC 协议呢?

在设计协议前，我们先梳理下要完成 RPC 通信的时候，在协议里面需要放哪些内容。

首先要想到的就是我们前面说的消息边界了，但 RPC 每次发请求发的大小都是不固定的，所以我们的协议必须能让接收方正确地读出不定长的内容。我们可以先固定一个长度（比如 4 个字节）用来保存整个请求数据大小，这样收到数据的时候，我们先读取固定长度的位置里面的值，值的大小就代表协议体的长度，接着再根据值的大小来读取协议体的数据，整个协议可以设计成这样：

不定长协议

但上面这种协议，只实现了正确的断句效果，在 RPC 里面还行不通。因为对于服务提供方来说，他是不知道这个协议体里面的二进制数据是通过哪种序列化方式生成的。如果不能知道调用方用的序列化方式，即使服务提供方还原出了正确的语义，也并不能把二进制还原成对象，那服务提供方收到这个数据后也就不能完成调用了。因此我们需要把序列化方式单独拿出来，类似协议长度一样用固定的长度存放，这些需要固定长度存放的参数我们可以统称为“协议头”，这样整个协议就会拆分成两部分：协议头和协议体。

在协议头里面，我们除了会放协议长度、序列化方式，还会放一些像协议标示、消息 ID、消息类型这样的参数，而协议体一般只放请求接口方法、请求的业务参数值和一些扩展属性。这样一个完整的 RPC 协议大概就出来了，协议头是由一堆固定的长度参数组成，而协议体是根据请求接口和参数构造的，长度属于可变的，具体协议如下图所示：

Bit Offset	0-15	16-47	48-63	64-71	72-79	80-87
0	魔术位	整体长度	消息ID	协议版本	消息类型	序列化方式
协议体	Payload					

定长协议

可扩展的协议

刚才讲的协议属于定长协议头，那也就是说往后就不能再往协议头里加新参数了，如果加参数就会导致线上兼容问题。举个具体例子，假设你设计了一个 88Bit 的协议头，其中协议长度占用 32bit，然后你为了加入新功能，在协议头里面加了 2bit，并且放到协议头的最后。升级后的应用，会用新的协议发出请求，然而没有升级的应用收到的请求后，还是按照 88bit 读取协议头，新加的 2 个 bit 会当作协议体前 2 个 bit 数据读出来，但原本的协议体最后 2 个 bit 会被丢弃了，这样就会导致协议体的数据是错的。

可能你会想：“那我把参数加在不定长的协议体里面行不行？而且刚才你也说了，协议体里面会放一些扩展属性。”

没错，协议体里面是可以加新的参数，但这里有一个关键点，就是协议体里面的内容都是经过序列化出来的，也就是说你要获取到你参数的值，就必须把整个协议体里面的数据经过反序列化出来。但在某些场景下，这样做的代价有点高啊！

比如说，服务提供方收到一个过期请求，这个过期是说服务提供方收到的这个请求的时间大于调用方发送的时间和配置的超时时间，既然已经过期，就没有必要接着处理，直接返回一个超时就好了。那要实现这个功能，就要在协议里面传递这个配置的超时时间，那如果之前协议里面没有加超时时间参数的话，我们现在把这个超时时间加到协议体里面是不是就有点重了呢？显然，会加重 CPU 的消耗。

所以为了保证能平滑地升级改造前后的协议，我们有必要设计一种支持可扩展的协议。其关键在于让协议头支持可扩展，扩展后协议头的长度就不能定长了。那要实现读取不定长的协议头里面的内容，在这之前肯定需要一个固定的地方读取长度，所以我们需要一个固定的写入协议头的长度。整体协议就变成了三部分内容：固定部分、协议头内容、协议体内容，前两部分我们还是可以统称为“协议头”，具体协议如下：

Bit Offset	0-15	16-47	48-63	64-71	72-79	80-87
0	魔术位	整体长度	头长度	协议版本	消息类型	序列化方式
88-103	消息ID					
不固定	协议头扩展字段					
协议体	Payload					

可扩展协议

最后，我想说，**设计一个简单的 RPC 协议并不难，难的就是怎么去设计一个可“升级”的协议。**不仅要让我们在扩展新特性的时候能做到向下兼容，而且要尽可能地减少资源损耗，所以我们协议的结构不仅要支持协议体的扩展，还要做到协议头也能扩展。上述这种设计方法来源于我多年的线上经验，可以说做好扩展性是至关重要的，期待这个协议模版能帮你避掉一些坑。

总结

我们人类区别于其他动物的一个很大原因，就是我们能够通过语言去沟通，用文字去沉淀文明，从而让我们能站在巨人的肩膀上成长，但为了保证我们记录的文字能够被其他人理解，我们必须通过符号去实现断句，否则就可能导致文字的意义被曲解，甚至闹出笑话。

在 RPC 里面，协议的作用就类似于文字中的符号，作为应用拆解请求消息的边界，保证二进制数据经过网络传输后，还能被正确地还原语义，避免调用方跟被调用方之间的“鸡同鸭讲”。

但我们在设计协议的时候，也不能只单纯考虑满足目前功能，还应该从更高的层次出发。就好比我们设计系统架构一样，我们需要保证设计出来的系统能够能很好地扩展，支持新增功能。

课后思考

好了，今天的内容就到这里，最后留一道思考题。今天我们讨论过 RPC 不直接用 HTTP 协议的一个原因是无法实现请求跟响应关联，每次请求都需要重新建立连接，响应完成后再关闭连接，所以我们要设计私有协议。那么在 RPC 里面，我们是怎么实现请求跟响应关联的呢？

欢迎留言和我分享你的思考，也欢迎你把文章分享给你的朋友，邀请他加入学习。我们下节课再见！

更多课程推荐

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇　01 | 核心原理：能否画张图解释下RPC的通信流程?

下一篇　03 | 序列化：对象怎么在网络中传输?

精选留言 (29)

写留言

楼下小黑哥
2020-02-18

以 Dubbo 为例，消费者发送请求时，使用 AtomicLong 自增，产生一个 消息 ID。由于 Dubbo 底层 IO 操作是异步的，Dubbo 发送请求之后，需要阻塞等待消费者返回信息。消费者会将消息 ID 保存到 Map 结构中,。为了保证请求响应可以一一对应，这就需要提供者返回的响应信息带上请求者消息 ID。 通过响应的消息 ID，通过 上面提到 Map 存储数据，就能找到对应的请求。...

展开

作者回复: 你好，非常正确。

1　16

小美
2020-02-19

老师好，您课后问题我有点不能理解。
我们http 请求一个资源不就对应一个返回。是一一对应的关系，为什么会有如何关联响应和请求的问题呢

展开

作者回复: rpc为了吞吐量，会异步并发发送请求，等待应答，所以需要知道哪个应答对应那个请求

3 6

(Kelen)
2020-02-19

hhtp现在已经支持长链接了啊

展开

作者回复: 可以，例如http2。目前性能不如tcp好。

1 4

jfwwlong
2020-02-21

你好，既然基于TCP优于HTTP，gRPC为什么选择基于HTTP2

作者回复: grpc基于http2，更容易跨语言支持。

3

旭杰
2020-02-18

调用方需要维护消息ID列表，然后和返回结果中的消息ID做匹配

作者回复: 是的

3

eason2017

2020-02-18

老师，你好。请求时带上消息id，响应时，响应体里面带上请求消息的id，这样可以进行关联，对吗？

作者回复: 很赞。异步场景用于区分应答消息。

1 2

忆水寒

2020-02-19

一般服务端会给每个客户端socket（或channel）绑定一个标识id，在注册中心可以通过id找到该socket（或channel），然后将数据发送出去。

展开

作者回复: 服务提供者会往注册中心注册，消费者从注册中心拿到提供者地址

1

听雨

2020-02-17

老师，关于http每次请求都要重新建立连接这一块不太理解，它不是支持长连接吗

作者回复: 你好，可以支持，应用http调用场景大部分都是短连接方式。

1 1

艾斯曼

2020-02-17

消息Id

展开

作者回复: 你好，艾斯曼。传统的rpc里面，请求的消息id用连接上的自增整数就行。用于标识请求消息，同时区分应答消息。

1

shen

2020-02-17

我的理解是在请求里面带一个请求id，然后响应对应的请求id，这样就建立了连接

作者回复: 你好，shen。请求id是用于标识请求，区分应答。

1

Harry
2020-02-24

老师，thrift 的协议中，好像就没有消息id ， thrift 不能做到请求和响应一一对应。
比如，多线程公用一个thrift 的client ， 并发请求server 端，client 会报相关的异常。

the day
2020-02-21

那么在 RPC 里面，我们是怎么实现请求跟响应关联的呢?
上下文匹配，简单明了!

展开

作者回复: 可以这么理解

Leon
2020-02-20

老师你好，对协议头加上序列化方式不太理解。如果某个rpc框架就支持protobuf，那么调用方和服务方都是基于protobuf，是不是就不需要加协议头里的序列化

展开

作者回复: 加上是为了灵活性。固定也可以，例如grpc就是protobuf。

vivi
2020-02-20

老师，“ HTTP 协议属于无状态协议，客户端无法对请求和响应进行关联。” 这应该是异步请求的情况下才会出现的吧。

展开

作者回复: 我们设计rpc的场景就是要大吞吐量，需要异步

Jackey
2020-02-20

答案就在题面上，老师的消息协议图里已经设计了消息ID了😃

作者回复: 看的仔细😁

婴
2020-02-19

对于课后思考里提到的“ RPC 不直接用 HTTP 协议的一个原因是无法实现请求跟响应关联”，我认为是有问题的，如果是同步请求，那么使用HTTP协议也是可以实现请求和相应关联的，只有异步请求才需要做关联，另外还需要说明这里的HTTP协议是指HTTP1.1。因为gRPC使用的是HTTP2.0协议，http2.0协议已经优化编码效率问题，而且HTTP2.0支持多路复用，不需要每次请求都需要重新建立连接，提高了连接的利用率。所以其实没必...

展开

作者回复: 在设计rpc的时候，是为了高性能和大吞吐量场景，基于tcp性能更快。压力不大场景，http可以满足。grpc做的也很好。

ZeroIce
2020-02-19

HTTP虽然是短连接，但是它不是支持长连接的？为什么不直接扩展HTTP请求支持长连接或者无状态->有状态，HTTP协议不可以扩展？

展开

作者回复: http协议构建在tcp上，性能慢。grpc构建在http2上的。

Tim Zhang
2020-02-18

对于服务提供方应用来说，他会从 TCP 通道里面收到很多的二进制数据 这里应该是服务消费方吧

作者回复: 服务提供方收到的请求二进制数据

谭方敏
2020-02-18

以我现在公司出来为例，会有个sequence id这个是唯一的，用于绑定请求与响应

作者回复: 是的，用于区分同一连接上的请求和响应

高源
2020-02-18

老师像今天讲的可扩展协议头这块，你可以简单举例子，或者demo，这样对于学习本次课程是个由简单到复杂过程，便于后期学习者积累，思考，突出实战，还有我们也是自定义协议包头包体socket交互，与rpc对比上那块还是不足效率和性能上

展开

作者回复: 具体的应用场景，例如在分布式跟踪的场景，需要把taceid传递下去，就可以利用可扩展头，写入traceid